Chapter 2: Lists and for loops

Author: Vincent Delecroix
Author: Mélodie Lapointe

In the first tutorial, you learnt how to call Sage functions and solve basic problems. To do more advanced computations you will need to learn a bit of programming. The aim of this tutorial is to introduce you to some basic Python programming: `list` that is one of the builtin Python data structures and `for` loops.

For reference you can have a look at [pydoc-datastructures] and [pydoc-controlflow].

Manipulating lists

In Python lists are constructed using square brackets (`[` and `]`) such as:

```
sage: l = [1, "two", -5.7, Primes()]
```

As you can notice, the data contained in a list is not necessarily homogeneous. To access elements of a list, one also uses square brackets:

```
sage: l[0]     # the first element
```

```
sage: l[2]     # the third elements
```

```
sage: l[-1]    # the last element
```

```
sage: l[4]     # this is out of range
```

An important function is `len` that returns the length of the list:

```
sage: len(l)    # number of elements in l
```

Given a list `l` and a non-negative integer `n` the operation `l * n` duplicates `n` times the list `l`:

```
sage: l * 3
```

Given two lists `l1` and `l2` the operation `l1 + l2` is the concatenation of `l1` and `l2`:

```
sage: [1, 2, 3] + [4, 5, 6]
```

Exercise 2.1

Construct the list that contains 1 time 1, 2 times 2, 3 times 3, ..., 10 times 10 (i.e. `[1, 2, 2, 3, 3, 3, ..., 10]`

```
sage: [x for x in range(1,11) for i in range(x)]
```

A for loop is a control flow that will allow to repeat some instructions for all elements of an *iterable* such as list.:

```
sage: l = [1, "hello", 3]
sage: for x in l:
....:     print(x)
```

You can have more than one instructions inside a for loop:

```
sage: l = [1, 2, 3, 4, 5]
sage: for x in l:
....:     y = 3 * x^ - 2
....:     print(y)
```

And it is possible to nest them:

```
sage: l1 = [1, 2, 3]
sage: l2 = [4, 7, 13]
sage: for x in l1:
....:     for y in l2:
....:         print(x, y, x^y)
```

Often you might want to perform a for loop for integers in a given range. To do so there are the functions `range` (from Python) and `srange` (from Sage).:

```
sage: for i in range(10):
....:     print(i)
```

```
sage: for i in srange(10):
....:     print(i)
```

The difference between `range` and `srange` is that `range` produces Python integer while `srange` produces Sage integers. You can compare:

```
sage: for i in range(10):
....:     print(i.is_prime())
```

```
sage: for i in srange(10):
....:     print(i.is_prime())
```

For both `range` and `srange` there are three possible syntaxes

- $range(stop)$: integers from 0 to $stop-1$ included
- $range(start, stop)$: integers from $start$ to $stop-1$
- $range(start, stop, step)$: integers from $start$ to $stop$ and with a difference of $step$ between each consecutive terms

Exercise 2.2

Write a for loop which prints the factorization for each integer from 1 to 100.

```
sage: for i in srange(1,101):
....:     show(i.factor())
```

Exercise 2.3

Write a for loop which prints the primitive of the functions $\sin(x)$, $\cos(x)$, $\tan(x)$, $\log(x)$, $\exp(x)$, $\sinh(x)$ and $\cosh(x)$.

```
sage: from sage.symbolic.integration.integral import indefinite_integral
....: x = var('x')
....: for i in [sin(x),cos(x),tan(x),log(x),exp(x),sinh(x),cosh(x)]:
....:     show(indefinite_integral(i,x))
```

Write a for loop which prints the derivates of the functions $f_n(x) = x^n$ for all values of n from 1 to 20.

```
sage: for i in range(1,21):
....:     show(indefinite_integral(x^i,x))
```

Exercise 2.4

Redo exercise 2.1 using a `for` loop

```
sage: l = []
....: for i in range(1,11):
....:     for j in range(i):
....:         l.append(i)
....: l
```

Repeat it with 100 instead of 10

```
sage: l = []
....: for i in range(1,11):
....:     for j in range(i):
....:         l.append(i)
```

Exercise 2.5

Write some code that displays the following figure using a loop

```
sage: print('*')
....: print('**')
....: print('***')
....: print('****')
....: print('*****')
....: print('******')
```

```
sage: for i in range(1,7):
....:     print('*'*i)
*
**
***
****
*****
******
```

Write some code that displays the following figure using a loop:

```
sage: print('*')
....: print('**')
....: print('***')
....: print('****')
....: print('*****')
....: print('******')
....: print('*****')
....: print('****')
....: print('***')
....: print('**')
....: print('*')
```

```
sage: for i in range(1,12):
....:     print('*'*min(i,12-i))
*
**
***
****
*****
******
```

```
*****
****
***
**
*
```

Write some code that displays the following figure using a loop:

```
sage: print('   *')
....: print('  ***')
....: print(' *****')
....: print('*******')
....: print(' *****')
....: print('  ***')
....: print('   *')
```

```
sage: for i in range(1,12,2):
....:     m = min(i,12-i)
....:     print (' '*(4-int((m+1)/2)) + ('*'*m))
   *
  ***
 *****
 *****
  ***
   *
```

Exercise 2.6

What is the value of

$$\sum_{k=1}^{20} k^k$$

```
sage: sum([x^x for x in range(1,21)])
106876212200059554303215024L
```

Solve $Eulerproblem48 < https://projecteuler.net/problem = 48 >$.

sage: str(sum([x^x for x in range(1,100)]))[-10:] '9027641920'

Note that Python `tuple` (constructed with parenthesis) and Python `string` (constructed with quotes `'` or `"`) behave the same with respect to many operations:

```
sage: t1 = (0, "hello", -2/3)

sage: t1[0]

sage: t1[-1]

sage: t1 * 3

sage: s1 = "I am in"
sage: s2 = "Saint-Flour"
sage: s1 + " " + s2
```

Exercise 2.7

Here is a list describing the size of 64 files:

```
sage: L = ["12K", "12K", "12K", "12K", "12K", "12K", "12K", "12K", "16K",
....: "16K", "16K", "20K", "20K", "20K", "24K", "24K", "24K", "24K", "24K",
....: "24K", "24K", "24K", "24K", "28K", "32K", "40K", "4K", "4K", "4K", "4K",
....: "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K",
....: "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K", "4K",
....: "4K", "4K", "4K", "4K", "8K", "8K", "8K", "8K", "8K", "8K"]
```

Compute the total size of the files, their average and the median

```
sage: L_int = [ZZ(x[:-1]) for x in L]
....: total_size = sum(L_int)
....: print(total_size)
696
```

```
sage: average = total_size/len(L)
....: print(average)
87/8
```

```
sage: median = sorted(L_int)[ZZ(len(L)/2)]
....: print(median)
8
```

(*hint*: to convert a string `s` like `"12"` into an integer simply do `ZZ(s)`)

How many files are there of each size? (you might want to look the available method on list object)

```
sage: size = {}
....: compteur = 0
....: valeur = L_int[0]
....: for i in sorted(L_int):
....:     if i == valeur:
....:         compteur += 1
....:     else:
....:         size[str(i)+'K'] = compteur
....:         compteur = 1
....:         valeur = i
....: print(size)
{'8K': 32, '28K': 9, '16K': 8, '32K': 1, '12K': 6, '40K': 1, '24K': 3, '4K': 0, '20K
```

```
sage: for i in set(L):
....:     print(i,L.count(i))
('8K', 6)
('28K', 1)
('16K', 3)
('32K', 1)
('12K', 8)
('40K', 1)
('24K', 9)
('4K', 32)
('20K', 3)
```

Construct the string that is the concatenation of each file size separated by a space

```
sage: s = ''
....: for i in L:
....:     s+= i + ' '
....: print(s)
12K 12K 12K 12K 12K 12K 12K 12K 16K 16K 16K 20K 20K 20K 24K 24K 24K 24K 24K 24K 24K
```

Can you construct the same string by increasing size?

```
sage: s = ''
```

```
....: for i in sorted(L,key=lambda y: ZZ(y[:-1])):
....:     s += i+ ' '
....: print s
4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K 4K
```

(*hint*: you might want to sort the list first... use tab-completion to find the appropriate list method to do this)

And by decreasing size?

```
sage: s = ''
....: for i in sorted(L,key=lambda y: ZZ(y[:-1]),reverse=True):
....:     s += i + ' '
....: print s
40K 32K 28K 24K 24K 24K 24K 24K 24K 24K 24K 24K 20K 20K 20K 16K 16K 16K 12K 12K 12K
```

(*hint*: have a look at the documentation of the method you used for sorting in the previous question)

Exercise 2.8

Let S_0 be the unit square with vertices $(0,0)$, $(1,0)$, $(1,1)$ and $(0,1)$. We define the square S_n obtained by joining the middle of each side of S_{n-1}.

Draw on the same pictures S_0, S_1, S_2, ... up to S_{10}.

```
sage: points = [(0,0),(1,0),(1,1),(0,1),(0,0)]
....: p = line2d(points)
....: for i in range(1,11):
....:     new_points = []
....:     for j in range(4):
....:         new_points.append(((points[j][0]+points[j+1][0])/2,(points[j][1]+point
....:     new_points.append(new_points[0])
....:     p += line2d(new_points)
....:     points = new_points
....: plot(p)
```

Do the same graphics starting from another quadrilateral which is not regular

```
sage: points = [(0,0),(2,0),(3,2),(0,1),(0,0)]
....: p = line2d(points)
....: for i in range(1,11):
....:     new_points = []
....:     for j in range(4):
....:         new_points.append(((points[j][0]+points[j+1][0])/2,(points[j][1]+point
....:     new_points.append(new_points[0])
....:     p += line2d(new_points)
....:     points = new_points
....: plot(p)
```

Do the same starting from a pentagon (five vertices)

```
sage: points = [(0,0),(1,0),(2,0.5),(1,1),(0,1),(0,0)]
....: p = line2d(points)
....: for i in range(1,11):
....:     new_points = []
....:     for j in range(5):
....:         new_points.append(((points[j][0]+points[j+1][0])/2,(points[j][1]+point
....:     new_points.append(new_points[0])
....:     p += line2d(new_points)
....:     points = new_points
....: plot(p)
```

Exercise 2.9

What does the following code?:

```
sage: x = 1.0
sage: for i in range(10):
....:     x = (x + 2.0 / x) / 2.0
```

What is the difference with:

```
sage: x = 1
sage: for i in range(10):
....:     x = (x + 2 / x) / 2
```

Exercise 2.10

Solve Euler problem 13. To simplify the exercise, the list of numbers has been created in the following cell:

```
sage: L = [37107287533902102798797998220837590246510135740250,
....: 46376937677490009712648124896970078050417018260538,
....: 74324986199524741059474233309513058123726617309629,
....: 91942213363574161572522430563301811072406154908250,
....: 23067588207539346171171980310421047513778063246676,
....: 89261670696623633820136378418383684178734361726757,
....: 28112879812849979408065481931592621691275889832738,
....: 44274228917432520321923589422876796487670272189318,
....: 47451445736001306439091167216856844588711603153276,
....: 70386486105843025439939619828917593665686757934951,
....: 62176457141856560629502157223196586755079324193331,
....: 64906352462741904929101432445813822663347944758178,
....: 92575867718337217661963751590579239728245598838407,
....: 58203565325359399008402633568948830189458628227828,
....: 80181199384826282014278194139940567587151170094390,
....: 35398664372827112653829987240784473053190104293586,
....: 86515506006295864861532075273371959191420517255829,
....: 71693888707715466499115593487603532921714970056938,
....: 54370070576826684624621495650076471787294438377604,
....: 53282654108756828443191190634694037855217779295145,
....: 36123272525000296071075082563815656710885258350721,
....: 45876576172410976447339110607218265236877223636045,
....: 17423706905851860660448207621209813287860733969412,
....: 81142660418086830619328460811191061556940512689692,
....: 51934325451728388641918047049293215058642563049483,
....: 62467221648435076201727918039944693004732956340691,
....: 15732444386908125794514089057706229429197107928209,
....: 55037687525678773091862540744969844508330393682126,
....: 18336384825330154686196124348767681297534375946515,
....: 80386287592878490201521685554828717201219257766954,
....: 78182833757993103614740356856449095527097864797581,
....: 16726320100436897842553539920931837441497806860984,
....: 48403098129077791799088218795327364475675590848030,
....: 87086987551392711854517078544161852424320693150332,
....: 59959406895756536782107074926966537676326235447210,
....: 69793950679652694742597709739166693763042633987085,
....: 41052684708299085211399427365734116182760315001271,
....: 65378607361501080857009149939512557028198746004375,
....: 35829035317434717326932123578154982629742552737307,
```

```
....: 94953759765105305946966067683156574377167401875275,
....: 88902802571733229619176668713819931811048770190271,
....: 25267680276078003013678680992525463401061632866526,
....: 36270218540497705585629946580636237993140746255962,
....: 24074486908231174977792365466257246923322810917141,
....: 91430288197103288597806669760892938638285025333403,
....: 34413065578016127815921815005561868836468420090470,
....: 23053081172816430487623791969842487255036638784583,
....: 11487696932154902810424020138335124462181441773470,
....: 63783299490636259666498587618221225225512486764533,
....: 67720186971698544312419572409913959008952310058822,
....: 95548255300263520781532296796249481641953868218774,
....: 76085327132285723110424803456124867697064507995236,
....: 37774242535411291684276865538926205024910326572967,
....: 23701913275725675285653248258265463092207058596522,
....: 29798860272258331913126375147341994889534765745501,
....: 18495701454879288984856827726077713721403798879715,
....: 38298203783031473527721580348144513491373226651381,
....: 34829543829199918180278916522431027392251122869539,
....: 40957953066405232632538044100059654939159879593635,
....: 29746152185502371307642255121183693803580388584903,
....: 41698116222072977186158236678424689157993532961922,
....: 62467957194401269043877107275048102390895523597457,
....: 23189706772547915061505504953922979530901129967519,
....: 86188088225875314529584099251203829009407770775672,
....: 11306739708304724483816533873502340845647058077308,
....: 82959174767140363198008187129011875491310547126581,
....: 97623331044818386269515456334926366572897563400500,
....: 42846280183517070527831839425882145521227251250327,
....: 55121603546981200581762165212827652751691296897789,
....: 32238195734329339946437501907836945765883352399886,
....: 75506164965184775180738168837861091527357929701337,
....: 62177842752192623401942399639168044983993173312731,
....: 32924185707147349566916674687634660915035914677504,
....: 99518671430235219628894890102423325116913619626622,
....: 73267460800591547471830798392868535206946944540724,
....: 76841822524674417161514036427982273348055556214818,
....: 97142617910342598647204516893989422179826088076852,
....: 87783646182799346313767754307809363333018982642090,
....: 10848802521674670883215120185883543223812876952786,
....: 71329612474782464538636993009049310363619763878039,
....: 62184073572399794223406235393808339651327408011116,
....: 66627891981488087797941876876144230030984490851411,
....: 60661826293682836764744779239180335110989069790714,
....: 85786944089552990653640447425576083659976645795096,
....: 66024396409905389607120198219976047599490197230297,
....: 64913982680032973156037120041377903785566085089252,
....: 16730939319872750275468906903707539413042652315011,
....: 94809377245048795150954100921645863754710598436791,
....: 78639167021187492431995700641917969777599028300699,
....: 15368713711936614952811305876380278410754449733078,
....: 40789923115535562561142322423255033685442488917353,
....: 44889911501440648020369068063960672322193204149535,
....: 41503128880339536053299340368006977710650566631954,
....: 81234880673210146739058568557934581403627822703280,
....: 82616570773948327592232845941706525094512325230608,
```

```
....: 22918802058777319719839450180888072429661980811197,
....: 77158542502016545090413245809786882778948721859617,
....: 72107838435069186155435662884062257473692284509516,
....: 20849603980134001723930671666823555245252804609722,
....: 53503534226472524250874054075591789781264330331690]
sage: str(sum(L))[:10]
'5537376230'
```

Modifying lists

Here are three ways to modify an already existing list `l`:

- `l[i] = j`: modify the element at position `i` to become `j`
- `l.append(j)`: append the element `j` at the end of `l`
- `l.pop()`: remove the last element of the list `l` and return it
- `l.extend(ll)`: add to `l` the content of the iterable `ll`

(there is also `l.insert(i, j)` that we will not use)

Exercise 2.11

What is the value of the list `l` at the end of the execution:

```
sage: l = [1, 2, 3]
sage: l.append(-1)
sage: l[1] = 7
```

(think about it before executing the lines)

What is the value of the list `l` at the end of the execution:

```
sage: l = [1]
sage: l.extend(l)
sage: l.extend(l)
```

(think about it before executing the lines)

Exercise 2.12

Given the following list of integers:

```
sage: l = [231, 442, 534, 667, 827, 314, 299, 351, 257, 688, 661, 123,
....: 567, 247, 151, 222, 605, 307]
```

Modify it so that each number at an even position is replaced by its double

```
sage: for i in range(0,len(l),2):
....:     l[i] = l[i]*2
....: print l
[462, 442, 1068, 667, 1654, 314, 598, 351, 514, 688, 1322, 123, 1134, 247, 302, 222,
```

Exercise 2.13

Let the following lists:

```
sage: t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
sage: t2 = ['January', 'February', 'March', 'April', 'May', 'June',
....: 'July', 'August', 'September', 'October', 'November', 'December']
```

Using `t1` and `t2` create a new list `t3` containing all elements of the two lists alternating them in such a way that each Month is followed by the corresponding number of days, that is, `['Janvier',31,'Février',28,'Mars',31, etc...]`

```
sage: t3 = []
....: for i in range(len(t1)):
....:     t3.append(t2[i])
....:     t3.append(t1[i])
....: print t3
['January', 31, 'February', 28, 'March', 31, 'April', 30, 'May', 31, 'June', 30, 'Ju
```

Exercise 2.14

Using the recurrence relation satisfied by the binomial numbers

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

compute the list $\binom{20}{0}, \binom{20}{1}, \ldots, \binom{20}{20}$. In order to do that you need to start from the list $[1]$ and design a loop that constructs successively $[1,1]$, then $[1,2,1]$, then $[1,3,3,1]$, etc. (It can be done using only one list.)

```
sage: binomial = [1]
....: for i in range(20):
....:     binomial.append(1)
....:     for j in range(i,0,-1):
....:         binomial[j] = binomial[j-1] + binomial[j]
....: print binomial
[1, 20, 190, 1140, 4845, 15504, 38760, 77520, 125970, 167960, 184756, 167960, 12597
```

Modify your loop to compute the Stirling numbers of the second kind that satisfies

$$S(n+1,k) = kS(n,k) + S(n,k-1)$$

with initial conditions $S(0,0) = 1$ and $S(n,0) = S(0,n) = 1$.

```
sage: stirling = [1]
....: for i in range(20):
....:     stirling.append(1)
....:     for j in range(i,0,-1):
....:          stirling[j] = stirling[j-1] + (j+1)*stirling[j]
....: print stirling
[1, 1048575, 1742343625, 181509070050, 3791262568401, 26585679462804, 8231095721494
```

List comprehension

List comprehension is a flexible way to build list. To build the list of squares n^2 from $n = 1$ to $n = 10$ one can do:

```
sage: l = [n^2 for n in srange(1, 11)]
....: print(l)
```

Exercise 2.15

Construct the same list of squares using a for loop and the method $.append()$.

```
sage: l = []
....: for i in srange(1,11):
....:     l.append(i^2)
....: print l
```

Exercise 2.16

Construct the list of powers of 5 for all values of exponents in the interval $[1,20]$

```
sage: [5^n for n in srange(1,20)]
```

Exercise 2.17

The Fibonacci sequence is defined by $F_0 = 0$, $F_1 = 1$ and for all $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$.

Make the list of the first 50 Fibonacci numbers F_n.

```
sage: fib = [0,1]
....: for i in range(50):
....:     fib.append(fib[-1] + fib[-2])
....: print fib
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181,
```

Using a for loop, print the values of $F_n^2 - F_{n-1}F_{n+1}$ for n between 1 and 48.

```
sage: for i in range(1,49):
....:     print fib[i]*fib[i] - fib[i-1]*fib[i+1]
```

Using a for loop, print the values of $F_{2n} - F_n^2 - F_{n-1}^2$.

```
sage: for i in range(1,25):
....:     print fib[2*i] - fib[i]*fib[i] - fib[i-1]*fib[i-1]
```

What do you remark? Could you prove it?

Exercise 2.18

What happens when you try to modify a string or a tuple?

```
sage: s = 'ssss'
....: s[1] = 'j'
sage: t = (1,2,3,4)
....: t[1] = 3
```

Exercise 2.19

Solve the Euler problem 18 and 67 about "maximum sum paths"

```
sage: L = [[75],[95,64],[17,47,82],[18,35,87,10],[20,4,82,47,65],[19,1,23,75,3,34],
sage: t = [0]
....: for ligne in L:
....:     for i in range(len(ligne),0,-1):
....:         if i == 1:
```

```
....:             t[i-1] = t[i-1] +  ligne[i-1]
....:         elif i == len(ligne):
....:             t.append(t[i-2] +  ligne[i-1])
....:         else:
....:             t[i-1] = ligne[i-1]+max(t[i-1],t[i-2])
....: print max(t)
1074
```

Exercise 2.20 (Champernowne constant)

The Champernowne constant is the real number obtained by concatenating all natural numbers in base 10

$$C = 0.12345678910111213141516171819202122 23\ldots$$

The *Champernowne word* is the sequence of digits. Using a for loop, construct the begining of the Champernowne word obtained by concatenating the integers from 1 to 100

```
sage: C = []
....: for i in srange(1,101):
....:     l = i.digits()
....:     l.reverse()
....:     C.extend(l)
....: print C
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 1, 1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9,
```

(*hint*: use the method `n.digits()` of Sage integers together with the method `l.extend(ll)` on lists)

Then solve Euler problem 48

```
sage: C = []
....: for i in srange(1,1000001):
....:     l = i.digits()
....:     l.reverse()
....:     C.extend(l)
....: solution = 1
....: for i in [10^i for i in srange(7)]:
....:     solution = solution*C[i-1]
....: print solution
210
```

Further comments

To learn more about iterators (that can be thought as "lazy lists") and in particular how to construct them, you can have a look at the Sage thematic tutorial on Comprehensions [sagett-comprehensions].

References

[pydoc-datastructures] https://docs.python.org/2.7/tutorial/datastructures.html [pydoc-controlflow] https://docs.python.org/2.7/tutorial/controlflow.html

[sagett-comprehensions] https://doc.sagemath.org/html/en/thematic_tutorials/tutorial-comprehensions.html